Towards Efficient Power Management in MapReduce: Investigation of CPU-Frequencies Scaling on Power Efficiency in Hadoop

Shadi Ibrahim, Diana Moise, Houssem-Eddine Chihoub, Alexandra Carpen-Amarie, Luc Bougé, Gabriel Antoniu

Towards Efficient Power Management in MapReduce: *Investigation of CPU-Frequencies Scaling on Power Efficiency in Hadoop*

Shadi Ibrahim[1]**, Diana Moise[2], Houssem-Eddine Chihoub[3], Alexandra Carpen-Amarie[4], Luc Bougé[5], and Gabriel Antoniu[1]

[1] Inria, Rennes Bretagne Atlantique Research Center, France
{shadi.ibrahim, gabriel.antoniu}@inria.fr
[2] InIT Cloud Computing Lab, ZHAW Winterthur, Switzerland
diana-maria.moise@zhaw.ch
[3] Inria, Sophia Antipolis Research Center, France
houssem-eddine.chihoub@inria.fr
[4] Vienna University of Technology, Austria
carpenamarie@par.tuwien.ac.at
[5] ENS Rennes / IRISA, France
luc.bouge@ens-rennes.fr

Abstract. With increasingly inexpensive cloud storage and increasingly powerful cloud processing, the cloud has rapidly become the environment to store and analyze data. Most of the large-scale data computations in the cloud heavily rely on the MapReduce paradigm and its Hadoop implementation. Nevertheless, this exponential growth in popularity has significantly impacted power consumption in cloud infrastructures. In this paper, we focus on MapReduce and we investigate the impact of dynamically scaling the frequency of compute nodes on the performance and energy consumption of a Hadoop cluster. To this end, a series of experiments are conducted to explore the implications of *Dynamic Voltage Frequency scaling* (DVFS) settings on power consumption in Hadoop-clusters. By adapting existing DVFS governors (i.e., *performance*, *powersave*, *ondemand*, *conservative* and *userspace*) in the Hadoop cluster, we observe significant variation in performance and power consumption of the cluster with different applications when applying these governors: the different DVFS settings are only sub-optimal for different MapReduce applications. Furthermore, our results reveal that the current CPU governors do not exactly reflect their design goal and may even become ineffective to manage the power consumption in Hadoop clusters. This study aims at providing more clear understanding of the interplay between performance and power management in Hadoop cluster and therefore offers useful insight into designing power-aware techniques for Hadoop systems.

Keywords: MapReduce, Hadoop, power management, DVFS, governors

** Corresponding author

1 Introduction

Power consumption has started to severely constrain the design and the way data-centers are operated. Power bills became a substantial part of the monetary cost for data-center operators. Hamilton [11] estimated that money spent on electrical power of servers and cooling units had exceeded 40 percent of total expenses of data-centers in 2008.

The surging costs of operating large data-centers have been mitigated by the advent of cloud computing, which allowed for better resource management, facilitated by the adoption of virtualization technologies. Nevertheless, overall energy consumption is continuously increasing as a result of the rapidly growing demand for computing resources. While various energy-saving mechanisms have been devised for large-scale infrastructures, not all of them are suitable in a cloud context, as they might impact the performance of the executed workloads. For instance, shutting down nodes to reduce power consumption may lead to aggressive virtual machine consolidation and resource over-provisioning, with dramatic effects on application performance. green cloud computing has thus emerged in an attempt to find a proper tradeoff between performance requirements and energy efficiency. To address this challenge, green clouds focus on the use of renewable energy sources, as well as on optimizing energy-saving mechanisms at the level of the data-center. Many research efforts have targeted power-saving techniques based on the Dynamic Voltage Frequency Scaling (DVFS) support in modern processors. In this paper, we aim at investigating the efficiency of such techniques in the context of large-scale data processing, which covers a major share of all cloud applications.

The most popular paradigm for data processing has been proposed by Google through their MapReduce model [6], which gained a wide adoption due to features including scalability, fault tolerance, and simplicity. Its most well-known open-source implementation, Hadoop [10], was designed to process hundreds of terabytes of data on thousands of cores at Yahoo!. As such large-scale deployments become a distinctive characteristic of cloud infrastructures, energy-efficient MapReduce is nowadays an essential concern in data-centers. Several studies have explored power saving in Hadoop clusters, through various techniques [2, 4].

MapReduce systems span over a multitude of computing nodes that are frequency and voltage-scalable. Our study, conducted on a Grid'5000 cluster [17], investigates the CPU-usage variation for three representative MapReduce benchmarks (*Pi*, *Grep* and *Sort*). As shown in Figure 1, the CPU load is high (more than 90%) during almost 75% of the job running time for the *Pi* application and is relatively high (more than 75%) only during 65% and 15% of the job running time for *Grep* and *Sort* jobs, respectively. Thus, there is a significant potential for reducing energy consumption by scaling down the CPU when the peak CPU performance is not required by the workload.

The contribution of this paper is to investigate such opportunities for optimizing energy consumption in Hadoop clusters. We rely on a series of experiments to explore the implications of DVFS settings on power consumption in Hadoop

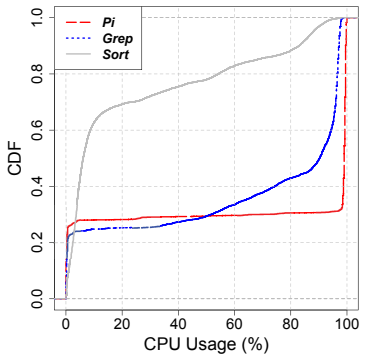


Fig. 1. CPU utilization when running *Pi*, *Grep* and *Sort* benchmarks with 7.5GB of data in a 15-node Hadoop cluster: *for the Pi and Grep applications, which represent CPU-intensive MapReduce applications, we observe that the CPU load is either high - more than 90% and 80% during 75% and 55% of the job running time - or low - less than 1% for 21% of the job running time, respectively. Conversely, for* Sort *application, a mostly I/O-intensive application, the CPU load has more variation.*

clusters. As DVFS research has reached a certain maturity, several CPU Frequency Scaling tools and governors have been proposed and implemented in the Linux kernel. For instance, governors such as *ondemand* or *performance* tune the CPU frequency to optimize application execution time, while *powersave* is designed to lower energy consumption.

We study the impact of different governors on Hadoop's performance and power efficiency. Interestingly, our experimental results report not only a noticeable variation of the power consumption and performance with different applications and under different governors, but also demonstrate the opportunity to achieve a better tradeoff between performance and power consumption.

The primary contributions of this paper are as follows:

1. It experimentally demonstrates that MapReduce applications experience variations in performance and power consumption under different CPU frequencies (similar to [32]) and also under different governors. A micro-analysis section is provided to explain this variation and its cause.
2. It illustrates in practice how the behavior of different governors influences the execution of MapReduce applications and how it shapes the performance of the entire cluster.

This study aims at providing a more clear understanding of the interplay between performance and power management in Hadoop clusters, with the purpose of deriving useful insights for designing power-aware techniques for Hadoop.

Paper Organization. The rest of this paper is organized as follows: Section 2 briefly presents Hadoop and the existing CPU power-management techniques. This section also discusses the related work. Section 3 describes an overview

of our methodologies, followed by the experimental results in Sections 4 and 5. Finally, we conclude the paper and propose our future work in Section 6.

2 Background and related work

In this section, we briefly introduce Hadoop and existing DVFS mechanisms. This section also presents related work on MapReduce energy consumption in data-centers and clouds.

2.1 Hadoop

Yahoo!'s Hadoop project [10] is a collection of various sub-projects for supporting scalable and reliable distributed computing. The two fundamental sub-projects feature a distributed file system (HDFS) and a Java-based open-source implementation of MapReduce through the Hadoop MapReduce framework. HDFS is a distributed file system that relies on a master/slave architecture to provide high-throughput access to application data [10]. The master server, called *namenode*, splits files into chunks and distributes them across the cluster with replication for fault tolerance. It holds all metadata information about stored files. The HDFS slaves are called *datanodes* and are designed to store data chunks, to serve read/write requests from clients and propagate replication tasks as directed by the *namenode*. Hadoop MapReduce is a software framework for distributed processing of large data sets on compute clusters. It runs on top of HDFS, thus collocating data storage with data processing. A centralized *Job Tracker* (JT) is responsible of: (a) querying the *namenode* for the block locations, (b) scheduling the tasks on *Task Trackers* (TT), based on the information retrieved from the *namenode*, and (c) monitoring the success and failures of the tasks.

2.2 Power Management at CPU Level

Modern processors offer the ability to tune the power mode of the CPU through the introduction of idle processor operating states (C-states) and CPU performance states (P-states). A C-state indicates whether the processor is currently active or not: processors in C_0 state are executing instructions while processors in higher C-states (C_i where $i = 1, 2$, *etc.*) are considered idle. Higher C-states reflect a deeper sleep mode, and thus increased power savings.

The P-states determine the processor frequencies and their associated voltage: Processors in the P_0 state run at the highest frequency and processors in the highest P-state run at the lowest frequency. The number of available P-states varies by processor type.

Dynamic Voltage Frequency Scaling (DVFS) is a commonly used technique that improves CPU utilization and power management by tuning the CPU frequency according to the current load. The ideal DVFS mechanism can instantaneously change the voltage/frequency values. Since the 2.6.10 version of the Linux kernel, there are five different governors available to dynamically scale the

CpuFreq Goevrnor	Goal	Short description	Downsides
Performance	Maximize Performance	Statically sets the CPU frequency to the highest available frequency	High power consumption
Powersave	Maximize power savings	Statically sets the CPU frequency to the lowest available frequency	Long response time
Ondemand	Power efficiency with reasonable performance	Dynamically adjusts the CPU frequency to the highest available frequency when the load is high and gradually degrades the CPU frequency when the load is low	Low performance/power saving benefits when the system switches between idle states and heavy load often
Conservative	Power efficiency with reasonable performance	Gradually upgrades the CPU frequency when the load is high and gradually degrades the CPU frequency when the load is low	Worse performance than Ondemand
Userspace	Support for user-defined frequencies	Statically sets the CPU frequency to a user-defined value	-

Table 1. CPU Governors

CPU frequency according to the CPU utilization. Each governor favors either performance or power efficiency, as shown in Table 1. More details can be found in [26]). Moreover, setting the governor to *userspace* allows users to use their own strategy in adjusting the CPU frequency. Additionally, modern CPUs provide a new feature called Turbo Boost which enhances the performance of a subset of a machine's cores by boosting their clock speed, while the rest of the available cores are in a sleep state.

2.3 Related Work

MapReduce has attracted much attention in the past few years [18]. Substantial research efforts have been dedicated to either adopting MapReduce in different environments such as multi-core [25], graphics processors (GPU)s [12], and virtual machines [15, 30] or to improving MapReduce performance through skew-handling [16, 21] and locality-execution [14, 33].

There have been several studies on evaluating and improving the MapReduce energy consumption in data-centers and clouds. Many of these studies focus on power-aware data-layout techniques [1, 19, 20, 23, 28, 29], which allow servers to be turned off without affecting data availability. GreenHDFS [19] separates the HDFS cluster into hot and cold zones and places the new or high-access data in the hot zone. Servers in the cold zone are transitioned to the power-saving mode and data are not replicated, thus only the server hosting the data will be woken up upon future access. Rabbit [1] is an energy-efficient distributed file system that maintains a primary replica on a small subset of always on nodes (active nodes). Remaining replicas are stored on a larger set of secondary nodes which are activated to scale up the performance or to tolerate primary failures. These data placement efforts could be combined with our approach to reduce the power consumption of powered servers. Instead of covering a set of nodes, Lang and Patel propose an all-in strategy (AIS) [22]. AIS saves energy in an all-or-nothing fashion: the entire MapReduce cluster is either on or off. All MapReduce jobs

are queued until a certain threshold is reached and then all the jobs are executed with full cluster utilization.

Some works consider energy saving for MapReduce in the cloud [2, 34]. Cardosa *et al.* [2] present virtual machines (VMs) replacement algorithms that co-allocate VMs with similar runtime on the same physical machine in a way that the available resources are highly utilized. Consequently, this maximizes the number of idle servers that can be deactivated to save energy. Chen *et al.* [5] analyze how MapReduce parameters affect energy efficiency and discuss the computation versus I/O tradeoffs when using data compression in MapReduce clusters in terms of energy efficiency [4]. Chen *et al.* [3] present the *Berkeley Energy Efficient MapReduce* (BEEMR), an energy efficient MapReduce workload manager motivated by empirical analysis of real-life MapReduce with Interactive Analysis (MIA) traces at Facebook. They show that interactive jobs operate on just a small fraction of the data, and thus can be served by a small pool of dedicated machines with full power, while the less time-sensitive jobs can run in a batch fashion on the rest of the cluster. Recently, Goiri *et al.* [9] present GreenHadoop, a MapReduce framework for a data-center powered by renewable green sources of energy (e.g. solar or wind) and the electrical grid (as a backup). GreenHadoop schedules MapReduce jobs when green energy is available and only uses brown energy to avoid time violations.

Closely related works focus on achieving power efficiency in Hadoop clusters by using DVFS [27, 32]. Li *et al.* [27] discuss the implications of temperature (machine heat) on performance and energy tradeoffs of MapReduce. Based on the the observation that higher temperature causes higher power consumption even with the same DVFS settings, they propose a temperature-aware power allocation (TAPA) that adjusts the CPUs frequencies according to their temperature. TAPA favors the maximum possible CPU frequency, thus maximizing computation capacity, without violating the power budget. Wirtz and Ge [32] compare the power consumption and the performance of Hadoop applications in three settings: (1) fixed frequencies, (2) setting the frequencies to maximum frequencies when executing the map or reduce otherwise minimum, and (3) performance-constraint frequency settings that tolerate some performance degradation while achieving better power consumption. Our work relies on the "Fine-grained" frequencies assignment, aiming to achieve the same performance while minimizing the power consumption.

Dynamic Voltage Frequency Scaling Techniques. There is a large body of work on techniques that control the DVFS mechanism for power-scalable PC cluster [7, 8, 13, 31, 24]. Some of these techniques control the CPU frequencies at runtime [13] and some scale the frequencies statically, based on extensive and expensive application profiling [7]. However, our approach differs from such works in the target applications (MapReduce applications).

3 Methodology Overview

The experimental investigation conducted in this paper focuses on exploring the implications of executing MapReduce applications in different DVFS settings. We conducted a series of experiments in order to assess the impact of various DVFS configurations on both power consumption and application performance. We further describe the experimental environment: the platform, deployment setup and used tools.

3.1 Platform

The experiments were carried out on the Grid'5000 [17] testbed. The Grid'5000 project provides the research community with a highly-configurable infrastructure that enables users to perform experiments at large scales. The platform is spread over 10 geographical sites located in France. For our experiments, we employed nodes belonging to the Nancy site on the Grid'5000. These nodes are outfitted with a 4-core Intel 2.53 GHz CPU and 16 GB of RAM. Intra-cluster communication is done through a 1 Gbps Ethernet network. It is worth mentioning that only 40 nodes of the Nancy site are equipped with power monitoring hardware consisting of 2 Power Distribution Units (PDUs), each hosting 20 outlets. Since each node is mapped to a specific outlet, we are able to acquire coarse and fine-grained power monitoring information using the Simple Network Management Protocol (SNMP). It is important to state that Grid'5000 allows us to create an isolated environment in order to have full control over the experiments and the obtained results.

3.2 Benchmarks

MapReduce applications are typically categorized as CPU-intensive, I/O bound, or both. For our analysis, we chose 3 applications that are commonly used for benchmarking MapReduce frameworks: *distributed grep*, *distributed sort* and *distributed pi*.

- **Distributed grep.** This application scans the input data in order to find the lines that match a specific pattern. The grep example can be easily expressed with MapReduce: the map function processes the input file line by line and matches each single line against the given pattern; if the matching is successful, then the line is emitted as intermediate data. The reduce function simply passes the intermediate data as final result.
- **Distributed sort.** The sort application consists in sorting key/value records based on key. With MapReduce, both the map and reduce functions are trivial computations, as they simply take the input data and emit it as output data. The sort MapReduce implementation takes advantage of the default optimizations performed by the framework that implicitly sorts both intermediate data and output data.

– **Distributed pi.** This benchmark estimates the value of pi based on sampling. The estimator first generates random points in a 1×1 area. The map phase checks whether each pair falls inside a 1-diameter circle; the reduce phase computes the ratio between the number of points inside the circle and the ones outside the circle. This ratio gives an estimate for the value of pi.

Of these 3 benchmarks, *pi* is purely CPU-intensive, while *grep* and *sort* are also I/O bound. However, *sort* is more data-intensive than *grep*, since it generates significantly more output data.

3.3 Hadoop deployment

On the testbed described in Section 3.1, we configured and deployed a Hadoop cluster using the Hadoop 1.0.4 stable version [10]. The Hadoop instance consists of the namenode, the jobtracker and the Hadoop client, each deployed on a dedicated machine, leaving 13 nodes to serve as both datanodes and tasktrackers. The tasktrackers were configured with 4 slots for running map tasks and 2 slots for executing reduce tasks. At the level of HDFS, we use the default chunk size of 64 MB and the default replication factor of 3 for the input and output data. In addition to facilitating the tolerance of faults, data replication favors local execution of mappers and minimizes the number of remote map executions. Prior to running the benchmarks, we generated 900 chunks of text (adding up to 56 GB) to feed the *grep* and *sort* applications. This input size results in fairly long execution time which allows us to thoroughly monitor power consumption information.

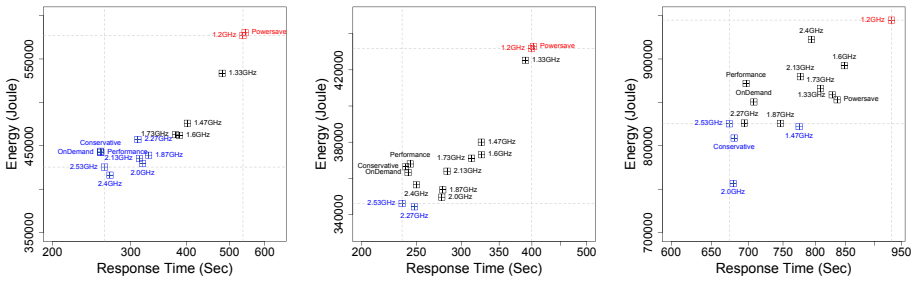
3.4 Dynamic Voltage Frequencies settings

The experiments involve running the benchmarks with various CPU settings and monitoring the power consumed by each node in this time frame. We distinguish a total of 15 scenarios corresponding to various values for CPU governors and frequencies. We were able to set the governor to conservative, on demand, performance, powersave, and userspace. With the governor set to userspace, we tune the CPU frequency to one of the following values: 1.2 GHz, 1.33 GHz, 1.47 GHz, 1.6 GHz, 1.73 GHz, 1.87 GHz, 2 GHz, 2.13 GHz, 2.27 GHz, 2.4 GHz, 2.53 GHz.

4 Macroscopic Analysis

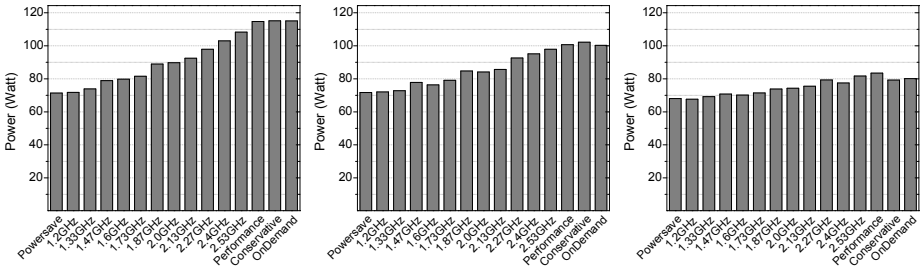
In this section, we provide a high-level analysis of the experimental results we obtained. Our goal is to study the impact of various *governors* or CPU frequencies on the performance of several classes of MapReduce applications.

Figure 2 depicts the completion time and the energy consumption of each *governor* for our three applications: *pi*, *grep* and *sort*. Each point on the graphs stands for the application runtime and the total energy consumption of the Hadoop cluster during its execution for a specific CPU frequency or *governor*.



(a) **Pi** (b) **Grep** (c) **Sort**

Fig. 2. Application runtime vs Energy consumption under various DVFS settings



(a) **Pi** (b) **Grep** (c) **Sort**

Fig. 3. Average Power consumption under various DVFS settings

We computed the total energy consumption for each application as the sum of the measured utilized power of each cluster node with a resolution of 1 second between measurements. In addition, Figure 3 displays a comparative view of the average power consumption of a job for each of the three applications and DVFS settings.

4.1 Performance analysis

The results show the job completion time increases as the employed CPU frequency decreases, for each of the three applications. In the case of the *pi* and *grep* applications, the runtime increases by 104% and 70%, respectively, when replacing the highest frequency, that is 2.53 GHz, with the lowest one, namely 1.2 GHz. The explanation for this behavior comes from the fact that the runtime of these two applications mostly accounts for computation, as they produce very little output data. Thus, the CPU performance has a significant impact on application execution time. The *sort* application is IO-bound, generating the same amount of output data as the input data. As in our experiments we employed an

input file of 900 chunks replicated 3 times, i.e. 56 GB of processed data and 168 GB of output data, *sort* spent a significant percentage of its execution time in reading data from and writing it to HDFS. Consequently, unlike *pi* and *grep*, the *sort* application exhibits a different behavior: reducing the CPU frequency from the highest to the lowest possible value only results in a 38% runtime increase.

These results are consistent with the CDF of the CPU usage depicted in Figure 1. *Pi* is a purely CPU-intensive application and consequently its CPU usage is the highest, amounting over 80% for most of the CPU frequencies and governors. At the other end of the spectrum, the IO-bound workload of *sort* is the main factor that accounts for an average CPU usage between 20% and 28%.

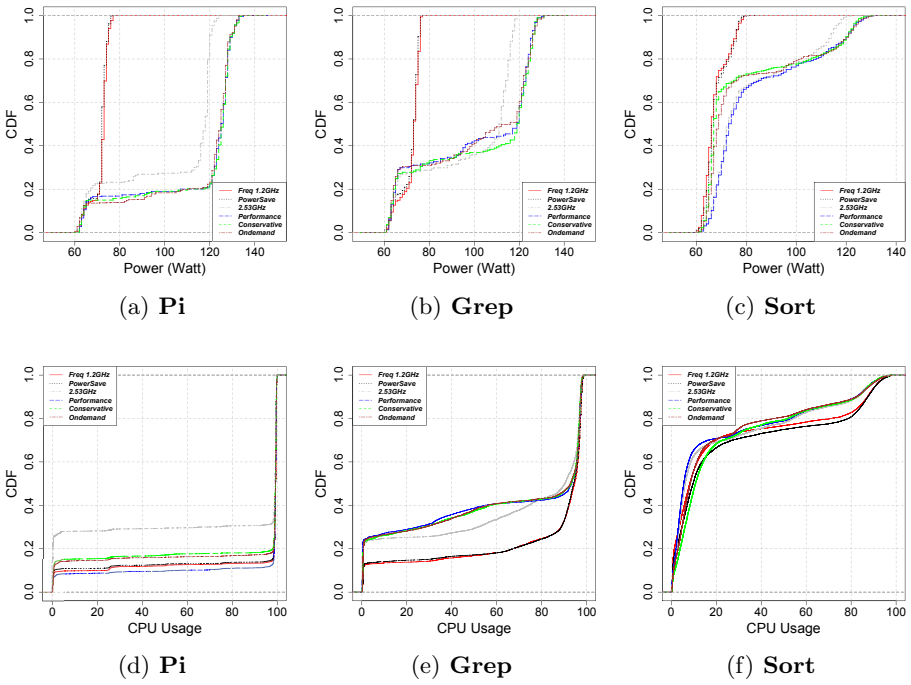
4.2 Energy consumption

The energy consumption on a Hadoop cluster depends on several parameters. One key factor is the CPU frequency, as low CPU frequencies also trigger low power consumption for a specific node. The application workload can however have an essential influence on the total energy utilized by the cluster. On the one hand, CPU-bound applications account for high CPU usage and thus for an increased energy consumption. Additionally, the application runtime directly impacts on the energy needed by the cluster, and thus attempts to improve application performance may result in better energy-efficiency. In this section we analyze the tradeoff between the aforementioned factors in the case of our three types of applications.

Figure 3(a) details the mean power consumption of a cluster node for each of the available fixed frequencies and all the governors, computed over the execution time of each application and the averaged across all cluster nodes. The average power consumption of a cluster node for *pi* is significantly lower for inferior CPU frequencies, as well as for the *powersave* governor. This observation would typically translate into an efficient total energy consumption at the level of the cluster for low frequencies. However, as Figure 2(a) demonstrates, the highest CPU frequency, that is 2.53 GHz, achieves the best results both in terms of performance and energy-efficiency. This behavior can be explained by analyzing the workload: *pi* is a CPU-intensive application, which can achieve 104% better performance by employing the highest CPU frequency, as shown in Section 4.1, whereas the average power consumption only increases by 48%.

The same trend can be noticed for the *grep* and *sort* applications. Nevertheless, the energy savings induced by using the highest available frequency proportionally decrease with the percentage of CPU usage of the application. Thus, as *sort* uses the least amount of CPU power, its runtime is not significantly impacted by reducing the CPU frequency and the total energy consumption of the application only increases by 15% between the highest and lowest CPU frequency values. The average power consumption for *sort* displayed in Figure 3(c) confirms this behavior, as the low CPU utilization at high CPU frequencies such as 2.53 GHz leads to only a 22% increase of the consumed power per node.

Consequently, the workload properties play an essential role in establishing the energy-consumption profile of an application. When the application runtime



(a) **Pi** (b) **Grep** (c) **Sort**

(d) **Pi** (e) **Grep** (f) **Sort**

Fig. 4. CDF of the average power consumption and CPU usage across nodes during application execution for various frequency and scaling policy settings.

predominantly accounts for CPU usage, the most power-consuming CPU settings can surprisingly trigger a better total energy efficiency. Accordingly, applications that feature both IO- and CPU-intensive phases, can benefit from adaptive CPU frequency policies, aiming at maximizing the CPU performance only during the computation stages of the application. Such policies can ensure a reduced energy consumption at the level of the application in two steps. First, for the CPU-intensive phases the total energy can be decreased by reducing the execution time, as it is the case for *Pi*. Second, the duration of the IO-bound phases is not dependent of the CPU frequency settings and therefore, low CPU frequencies can be used to save energy.

5 Microscopic Analysis

In this section, we present a detailed comparative discussion of various CPU frequencies and policies and we explain their effects on the total energy consumption of applications.

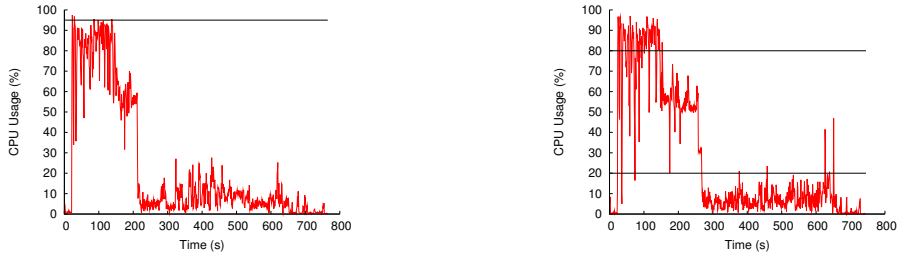
5.1 Dynamic frequency scaling

The highest frequency that can be statically configured on the cluster nodes is 2.53 GHz, this being also the default frequency employed by the operating system. We consider this frequency as the baseline against which we study the two dynamic governors, as it provides the default application performance that can be achieved by the given machines. Both the *ondemand* and *conservative* governors are designed to dynamically adjust the CPU frequency to favour either performance or energy consumption.

CPU-bound applications. When running the *pi* benchmark, both governors achieve slightly better performance than the default CPU frequency, as shown in Figure 2(a). This behavior can be explained by the fact that both governors attempt to increase the employed CPU frequency as much as possible when dealing with a CPU-intensive workload, as it is the case for *pi*.

Figure 4 presents the cumulative distribution function (CDF) of the average power consumption and average CPU usage during benchmark execution across the cluster nodes, for the various CPU frequency settings and in each of the three scenarios we analyzed. The CPU utilization is higher than 98% for more than 80% of the execution time (as shown in Figure 4(d)) for both governors. The CPU-bound nature of the *pi* application accounts for these values, as well as for the identical behavior of the two governors. Thus, as the CPU usage increases to almost 100% when the application is executed, the *conservative* and *ondemand* governors switch to the highest available frequency and do not shift back to lower frequencies until the job has finished and the CPU is released.

Interestingly, Figure 2(a) shows that the total energy consumption of the *pi* benchmark for the default frequency does not match the one corresponding to the performance-oriented governors, in spite of their similar execution times. The explanation lies in the processor ability to use the Turbo Boost capability when configured to employ an adaptive governor instead of a fixed frequency. The power consumption CDF for *pi* in Figure 4(a) shows the used power for the default CPU frequency is almost constant to 120 Watts for 80% of the job. As the CPU usage does not decrease during the execution of the *pi* application, this value represents the maximum power that the node can consume within a fixed frequency setting. However, the power consumption achieved by the two governors exceeds that of the default frequency, as emphasized by the *pi* CDF in Figure 4(a). This outcome is only possible if the governors take advantage of the CPU Turbo Boost capability, that is they employ a frequency higher than 2.53 GHz and in turn consume an increased amount of energy. Figure 6 provides an insight into the percentage of the job execution time spent by each governor with an enabled Turbo feature for each of the three applications. As previously anticipated, in the case of *pi*, the *conservative* governor invokes Turbo frequencies for 70% of the total time, while the *ondemand* governor requires Turbo for 65% of the running time.



(a) **Ondemand** (b) **Conservative**

Fig. 5. CPU usage on a Hadoop datanode during the execution of the *sort* application.

IO-bound applications. However, the *conservative* and *ondemand* governors behave differently for the *sort* application. As Figure 2 shows, when using the *ondemand* governor, Hadoop requires more time to sort the input data than when it is configured with the *conservative* governor. This longer running time also results in higher power consumption. To better understand how these two governors function, we analyze the CPU usage as a function of execution time on a single datanode, during the *sort* benchmark (Figure 5). Both governors start the execution at the default frequency (2.53 GHz), but they adjust the CPU frequency according to the CPU usage and how it compares to predefined thresholds.

The *ondemand* governor uses as threshold a default value of 95%: when the CPU usage is greater than 95%, the CPU frequency is increased to the highest available frequency, i.e 2.53 GHz; if the CPU usage is less than this value, the governor gradually decreases the frequency to lower values. In the case of the *sort* benchmark, this policy allows Hadoop to run at the highest frequency in some points corresponding to the CPU usage peaks above the 95% threshold (Figure 5(a)). Nevertheless, the rest of the CPU-intensive phase of the *sort* application is executed at lower CPU frequencies, since the CPU usage during this phase is less than 95%. The *conservative* governor employs two thresholds for tuning the CPU frequency: an up-threshold set to 80% and a down-threshold of 20%. The frequency is progressively increased and decreased by comparing the system usage to the two thresholds: CPU usage peaks above the up-threshold result in upgrading the frequency to the next available value; when the usage goes below the down-threshold, the CPU switches to the next lower frequency. Figure 5(b) shows that the computational-intensive phase of the *sort* benchmark exhibits CPU usage peaks greater than 80%. This enables the conservative governor to keep the CPU at the highest frequency of 2.53 GHz during most of this phase. Also, the down-threshold allows the I/O-bound part of the application to be executed at low CPU frequencies.

The internal implementation of the two governors is also responsible for the overall variation in performance and consumed energy between the static

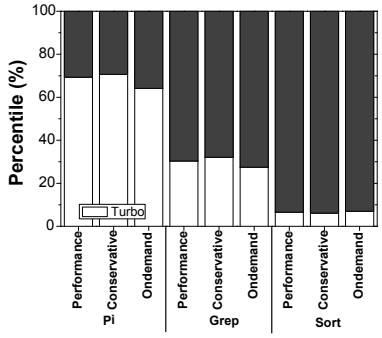


Fig. 6. Turbo Boost: The total usage of the Turbo feature for the duration of the application execution.

2.53 GHz setting and the dynamic governors detailed in Figure 2(c). Thus, in the case of *sort*, the *conservative* governor achieves a better runtime than *ondemand*, despite the fact that the latter governor should favour performance. While the improvement accounts for less than 5% of the execution time of the *ondemand* governor, it can be explained by the fact that the *conservative* governor spends more time at the highest frequency setting, speeding up the computational-intensive phases of *sort*. As most peaks in the CPU load do not reach the 95% threshold required by the *ondemand* governor, it cannot take advantage of the highest available frequency, leading to a worse application runtime. Energy-wise, this behaviour translates into a total energy gain when using the fixed 2.53 GHz frequency setting, on account on the longer execution time triggered by the *ondemand* governor. The *conservative* governor is in this case the best choice for saving energy, as it enables the application to take advantage of both high and low frequency settings, reducing the execution time of CPU-intensive phases and decreasing energy consumption during IO-intensive ones.

5.2 Statically-configured frequencies

In this section we focus on the *performance* and *powersave* governors, which set the CPU to a fixed frequency, either the maximum available one, that is 2.53 GHz or the minimum 1.2 GHz, respectively. While they should exhibit similar behaviour with the fixed frequency, the *performance* governor features an interesting capability, that is to use Turbo Boost when executing heavy loads.

As far as the *pi* application is concerned, both *performance* and the fixed 2.53 GHz frequency setting deliver relatively similar running times. The Turbo feature allows the *performance* governor to go past the 2.53 GHz CPU frequency for 70% of the job execution time and thus consume more power, as confirmed by the CDF in Figure 4(a). A notable side effect is that the usage of the *performance* governor is less efficient from an energy standpoint (Figure 2(a)). Figure 4(b) shows a similar behaviour in the case of *grep*, as it exhibits a sufficiently similar workload. Consequently, for CPU-bound applications, performance-oriented

governors provide a convenient alternative over a fixed frequency, when the user tends to favor performance. To achieve energy savings without significantly sacrificing execution time, the default kernel setting, the fixed maximum frequency still provides the best alternative. As for *sort*, the generated CPU peaks account for a limited usage of the Turbo CPU feature, as detailed in Figure 6. As a result, *sort* does not benefit from the *performance* governor in terms of energy, providing a better performance-energy tradeoff when using the 2.53 GHz setting or the *conservative* governor.

6 Summary and Future Work

Energy efficiency has started to severely constrain the design and the way data-centers are operated, becoming a key research direction in the development of cloud infrastructures. As processing huge amounts of data is a typical task assigned to large-scale cloud platforms, several studies have been dedicated to improving power consumption for data-intensive cloud applications. In this study, we focus on MapReduce and we investigate the impact of dynamically scaling the frequency of compute nodes on the performance and energy consumption of a Hadoop cluster. We provide a detailed evaluation of a set of representative MapReduce workloads, highlighting a significant variation in both the performance and power consumption of the applications with different governors.

Furthermore, our results reveal that the current CPU governors do not exactly reflect their design goal and may even become ineffective at improving power consumption for Hadoop clusters. In addition, we unveil the correlations between the power efficiency of a Hadoop deployment, application performance and power-management mechanisms, such as DVFS or Turbo capabilities. We believe the insights drawn from this paper can serve as guidelines for efficiently deploying and executing data-intensive applications in large-scale data-centers.

As future work, we plan to extend our empirical evaluation for a wider diversity of MapReduce applications, such as scientific applications and the more complex pipeline MapReduce applications, and for various platforms (e.g., virtualized data-centers). In addition, we intend to explore different techniques and approaches to optimize power management in Hadoop clusters. As a first step, we are currently investigating the possibility of building dynamic frequency tuning tools specifically tailored to match MapReduce application types and execution stages.

7 Acknowledgments

This work is supported by the ANR MapReduce grant (ANR-10-SEGI-001) and the Héméra INRIA Large Wingspan-Project (see http://www.grid5000.fr/mediawiki/index.php/Hemera).

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including

CNRS, RENATER and several Universities as well as other organizations (see http://www.grid5000.fr/).

References

1. Hrishikesh Amur, James Cipar, Varun Gupta, Gregory R. Ganger, Michael A. Kozuch, and Karsten Schwan. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 217–228, New York, NY, USA, 2010. ACM.
2. Michael Cardosa, Aameek Singh, Himabindu Pucha, and Abhishek Chandra. Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 251–258, Washington, DC, USA, 2011.
3. Yanpei Chen, Sara Alspaugh, Dhruba Borthakur, and Randy Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*, Bern, Switzerland, 2012.
4. Yanpei Chen, Archana Ganapathi, and Randy H. Katz. To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*, Green Networking '10, pages 23–28, New York, NY, USA, 2010. ACM.
5. Yanpei Chen, Laura Keys, and Randy H. Katz. Towards energy efficient mapreduce. Technical Report UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009.
6. Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
7. Vincent W. Freeh and David K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 164–173, 2005.
8. Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):658–671, May 2010.
9. Inigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenhadoop: Leveraging green energy in data-processing frameworks. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*, Bern, Switzerland, 2012.
10. The Apache Hadoop Project. http://www.hadoop.org, 2014.
11. James Hamilton. Cost of Power in Large-Scale Data Centers. http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx, 2008.
12. Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang. Mars: a mapreduce framework on graphics processors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 260–269, Toronto, Ontario, Canada, 2008.
13. Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 1–, Washington, DC, USA, 2005. IEEE Computer Society.

14. Shadi Ibrahim, Hai Jin, Lu Lu, Bingsheng He, Gabirel Antoniu, and Song Wu. Maestro: Replica-aware map scheduling for mapreduce. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, pages 59–72, Ottawa, Canada, 2012.
15. Shadi Ibrahim, Hai Jin, Lu Lu, Li Qi, Song Wu, and Xuanhua Shi. Evaluating mapreduce on virtual machines: The hadoop case. In *Proceedings of the 1st International Conference on Cloud Computing (CLOUDCOM'09)*, pages 519–528, Beijing, China, 2009.
16. Shadi Ibrahim, Hai Jin, Lu Lu, Song Wu, Bingsheng He, and Li Qi. Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CLOUDCOM'10)*, pages 17–24, Indianapolis, USA, 2010.
17. Yvon Jégou, Stephane Lantéri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Touche Iréa. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.
18. Hai Jin, Shadi Ibrahim, Li Qi, Haijun Cao, Song Wu, and Xuanhua. Shi. The mapreduce programming model and implementations. *Cloud computing: Principles and Paradigms*, pages 373–390, January 2011.
19. Rini T. Kaushik and Milind Bhandarkar. Greenhdfs: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster. In *Proceedings of the 2010 international conference on Power aware computing and systems*, HotPower'10, pages 1–9, Berkeley, CA, USA, 2010. USENIX Association.
20. Jinoh Kim, Jerry Chou, and Doron Rotem. Energy proportionality and performance in data parallel computing clusters. In *Proceedings of the 23rd international conference on Scientific and statistical database management*, SSDBM'11, pages 414–431, Berlin, Heidelberg, 2011. Springer-Verlag.
21. YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 75–86, Indianapolis, Indiana, USA, 2010.
22. Willis Lang and Jignesh M. Patel. Energy management for mapreduce clusters. *Proc. VLDB Endow.*, 3(1-2):129–139, September 2010.
23. Jacob Leverich and Christos Kozyrakis. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, March 2010.
24. Yousri Mhedheb, Foued Jrad, Jie Tao, Jiaqi Zhao, Joanna Kolodziej, and Achim Streit. Load and thermal-aware vm scheduling on the cloud. In *Proceedings of the 13th International Conference (ICA3PP 2013)*, pages 137–150, Vietri sul Mare, Italy, 2013.
25. Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture (HPCA-13)*, pages 13–24, Phoenix, Arizona, USA, 2007.
26. Redhat. Using CPUfreq Governors. `https://access.redhat.com/site/documentation/en-US/Red\_Hat\_Enterprise\_Linux/6/html/Power\_Management\_Guide/cpufreq\_governors.html`, 2014.
27. Li Shen, T Abdelzaher, and Mindi Yuan. Tapa: Temperature aware power allocation in data center with map-reduce. In *Proceedings of 2011 International Green*

Computing Conference and Workshops (IGCC'11), Green Networking '10, pages 1–8, New York, NY, USA, 2011. IEEE.

28. Eno Thereska, Austin Donnelly, and Dushyanth Narayanan. Sierra: practical power-proportionality for data center storage. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 169–182, New York, NY, USA, 2011. ACM.
29. Nedeljko Vasić, Martin Barisits, Vincent Salzgeber, and Dejan Kostic. Making cluster applications energy-aware. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACDC '09, pages 37–42, New York, NY, USA, 2009. ACM.
30. Lizhe Wang, Jie Tao, Rajiv Ranjan, Holger Marten, Achim Streit, Jingying Chen, and Dan Chen. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. *Future Gener. Comput. Syst.*, 29(3):739–750, March 2013.
31. Xiaorui Wang, Xing Fu, Xue Liu, and Zonghua Gu. Power-aware cpu utilization control for distributed real-time systems. In *Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, RTAS '09, pages 233–242. IEEE Computer Society, 2009.
32. Thomas Wirtz and Rong Ge. Improving mapreduce energy efficiency for computation intensive workloads. In *Proceedings of 2011 International Green Computing Conference and Workshops* (IGCC'11), Green Networking '10, pages 1–8, New York, NY, USA, 2011. IEEE.
33. Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th ACM European Conference on Computer Systems (EuroSys'10)*, pages 265–278, Paris, France, 2010.
34. Nan Zhu, Lei Rao, Xue Liu, Jie Liu, and Haibin Guan. Taming power peaks in mapreduce clusters. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 416–417, New York, NY, USA, 2011. ACM.